“Jožef Stefan” Institute

Department of Communication Systems

SensorLab

SensorLab VESNA open source development environment setup guide for Windows based development

version 0.10

by Marko Mihelin

July 24, 2014

1 Introduction

The setup and usage of the open source development environment for the VESNA platform is described in this document. This manual is for Windows based development and has been successfully tested on Windows 7 and 8, 32bit and 64bit versions.

1.1 Overview of the installation process

1. Install Cygwin
2. Install CodeBench tool-chain
3. Install OpenOCD
4. Install ST Flash Loader Demonstrator (Optional)
5. Install Eclipse
6. Install Eclipse plug-ins (tool-chain support, debugging support, ...)
7. Connect the debugger interface and install the drivers
8. Download VESNA repository, build and debug

Contents

1 Introduction **1**
1.1 Overview of the installation process . . . 1

2 Install Cygwin **5**

3 Install Sourcery CodeBench tool-chain **7**

4 Install OpenOCD **9**

5 Install ST Flash Loader Demonstrator (Optional) **9**

6 Install Eclipse IDE **9**
6.1 Install Eclipse . . . 9
6.2 Install the GNU ARM Development support . . . 11
6.3 Install the Zylin CDT plug-in . . . 12
6.4 Install the EmbSys peripheral register view plug-in . . . 13
6.5 Configure Eclipse Git client . . . 16

7 Connection of the debugger interface and installation of drivers **17**

8 Download VESNA repository, Build and Debug **20**
8.1 Download VESNA repository from Github . . . 20
8.2 Import project . . . 24
8.3 Configure the project . . . 27
8.4 Compile the source . . . 29

9 Upload the project and start debug **33**

List of Figures

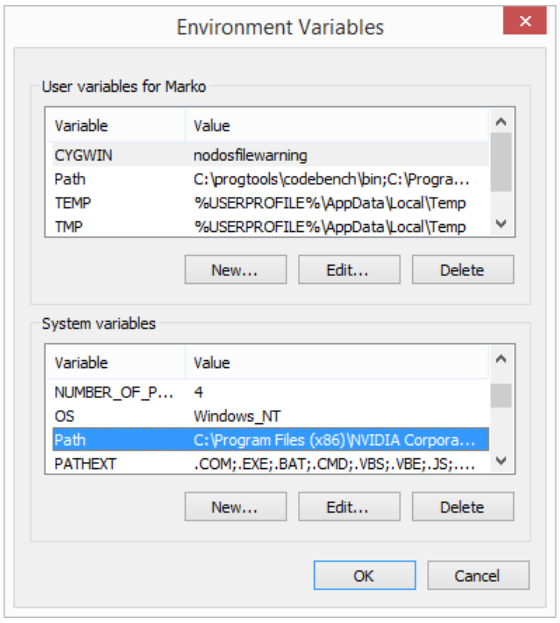
1 Selecting the PATH environment variable 5
2 Add new path to PATH variable 6
3 Tool-chain installation welcome window 7
4 Modify PATH for all users 8
5 Eclipse workspace prompt 10
6 Eclipse welcome window 10
7 Install GNU ARM development support 11
8 Install the Zylin Embedded CDT 12
9 EmbSys Register View plug-in installation 13
10 EmbSys Register View setup 14
11 EmbSys Register view (red square) 15
12 Eclipse Git client configuration 16
13 Device Manager when STLink is connected for the first time 17
14 Select method to find driver for STLink 18
15 Browse to driver location 18
16 Driver installation complete 19
17 Git perspective in Eclipse 20
18 Git repository location and credentials 21
19 Git repository branch selection 22
20 Git repository local destination 23
21 Project import wizard selection 24
22 New project type selection 25
23 Import existing code window 26
24 Project tool-chain settings 27
25 Documentation tool settings 28
26 C/C++ perspective demo project 29
27 Create make target 30
28 ”Make Target” tab (in red square) 31
29 Building the code 32
30 External tool selection 33
31 External tool running 34
32 Debugger configuration 35
33 Prompt to switch to ”Debug” perspective 36
34 Debug perspective, yellow frame: debug controls, blue frame: running debug tools and current stack position, red frame: local variables and green frame: current position in code 37

List of Abbreviations

IDE	Integrated Development Environment
SNC	Sensor Node Core module
MRAM	Magnetoresistive random-access memory

2 Install Cygwin

Cygwin is a Linux-like environment for Windows. It is needed because the build process of the projects require some Linux specific commands that Cygwin provides. You can download it from `http://cygwin.com/`. The version at the time of writing was 1.7.30. You can install it with default options. Make sure you add the *"C:\install\dir\cygwin\bin"* to your system PATH. You do this by going to Control Panel and clicking on System. Got to *Advanced system settings* → *Environment Variables*.

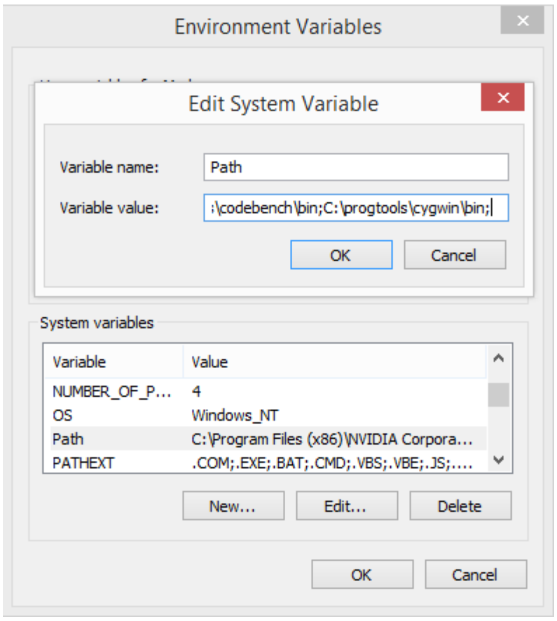


Variable	Value
CYGWIN	nodosfilewarning
Path	C:\progtools\codebench\bin;C:\Progra...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
NUMBER_OF_P...	4
OS	Windows_NT
Path	C:\Program Files (x86)\NVIDIA Corpora...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

Figure 1: Selecting the PATH environment variable

Now find the variable *PATH* in the *System Variables* window, select it and click edit (Figure 1).



Variable name: Path

Variable value: :\codebench\bin;C:\progtools\cygwin\bin;

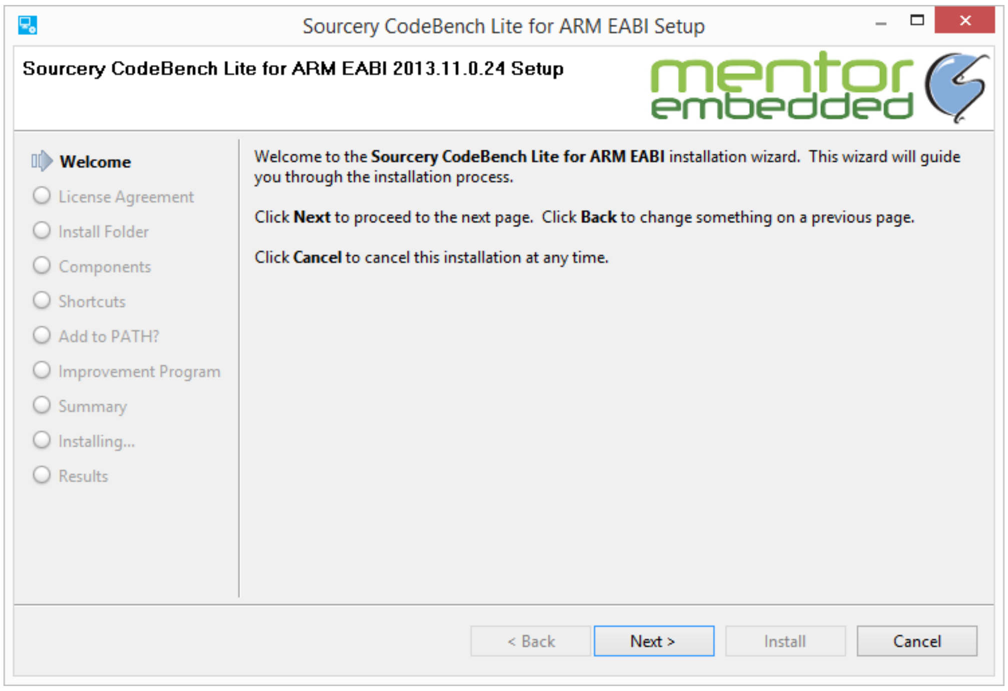
Variable	Value
NUMBER_OF_P...	4
OS	Windows_NT
Path	C:\Program Files (x86)\NVIDIA Corpora...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

Figure 2: Add new path to PATH variable

At the end of the string of paths in *"Variable value"* field add a semicolon (;) and the path to Cygwin install directory like in Figure 2. Make sure not to delete any preexisting paths. Click *OK*.

3 Install Sourcery CodeBench tool-chain

The tool-chain contains the compiler linker and debugger for ARMv7M software development. The tool-chain can be download from `http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/arm-eabi`. You have to provide your name and e-mail. A download link will be sent to you via e-mail. Select the *"Sourcery CodeBench Lite 2013.11-24"* release. Newer releases may also work but were not tested at the time of writing. Start the installation.



Sourcery CodeBench Lite for ARM EABI 2013.11.0.24 Setup

Welcome to the **Sourcery CodeBench Lite for ARM EABI** installation wizard. This wizard will guide you through the installation process.

Click **Next** to proceed to the next page. Click **Back** to change something on a previous page.

Click **Cancel** to cancel this installation at any time.

Figure 3: Tool-chain installation welcome window

Chose an install path with no spaces (spaces can cause problems with Eclipse, and the debugger) (Figure 3).

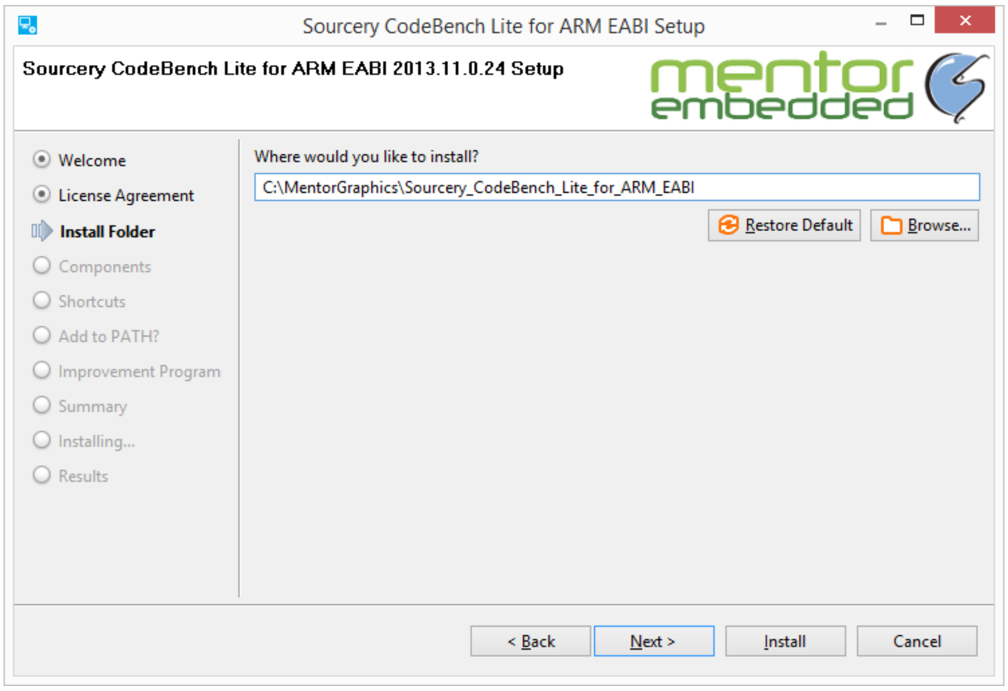

Sourcery CodeBench Lite for ARM EABI Setup
Sourcery CodeBench Lite for ARM EABI 2013.11.0.24 Setup
mentor embedded
Welcome
License Agreement
Install Folder
Components
Shortcuts
Add to PATH?
Improvement Program
Summary
Installing...
Results
Where would you like to install?
C:\MentorGraphics\Sourcery_CodeBench_Lite_for_ARM_EABI
Restore Default
Browse...
< Back
Next >
Install
Cancel

Figure 4: Modify PATH for all users

Select *"Modify PATH for current users"* (Figure 4). Finish the installation. Make sure that path to the tool-chain was added to *System PATH*.

4 Install OpenOCD

OpenOCD is the interface between the hardware JTAG and the debugger, the current stable version is 0.8.0. It can be build form source or a precompiled binary can be downloaded from `http://www.freddiechopin.info/en/download/category/4-openocd`. Select the 0.8.0 version. Unzip the folder where you want to place OpenOCD. Make sure that the installation path does not contain any spaces or non-US characters. Then go to the unzip folder and rename it form *OpenOCD-0.8.0* to *OpenOCD*. Go to sub-folder *bin* or *bin-x64* (depending on your system, 32bit or 64bit Windows) and rename *openocd*.exe* to *openocd.exe*. Add the *bin* or *bin-x64* (again, depending on your system) folder to your *System PATH*.

5 Install ST Flash Loader Demonstrator (Optional)

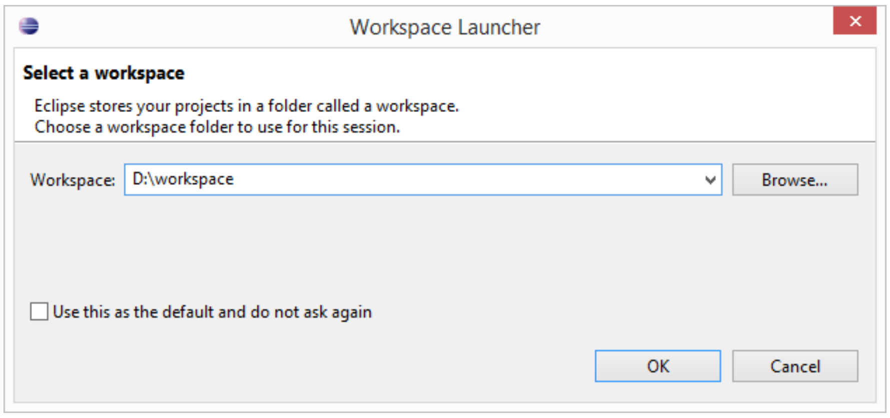
The Flash Loader can upload software to target microcontroller using only a com port. It can upload binary .bin or .s19 images. This is useful for programing SNC when JTAG can not be used, for instance when using the on-board MRAM. The Flash loader can be downloaded from ST ST Flash Loader Demonstartor. To initialize the on-board bootloader you must push and hold the button on the SNC for more then 5 seconds. Then use the Flash loader to connect and upload the software.

6 Install Eclipse IDE

Before installing Eclipse you must have the latest 32bit *Java Run-time Environment* installed. You can find it at `https://www.java.com/en/`.

6.1 Install Eclipse

Eclipse can be downloaded from `http://www.eclipse.org/downloads/`. The newest release at the time of writing was Kepler (4.3). Newer releases may also work but are not yet tested. Select the *"Eclipse IDE for C/C++ Developers"* 32bit version and download it. Eclipse does not need installation, just unzip in a desired location and run it. Start Eclipse now. You are prompted for workspace location (Figure 5), this is were the settings are stored. You can select the default location. Make sure the path name does not contain any non-US characters and spaces.


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\workspace

Figure 5: Eclipse workspace prompt

You are presented with a welcome window (Figure 6).

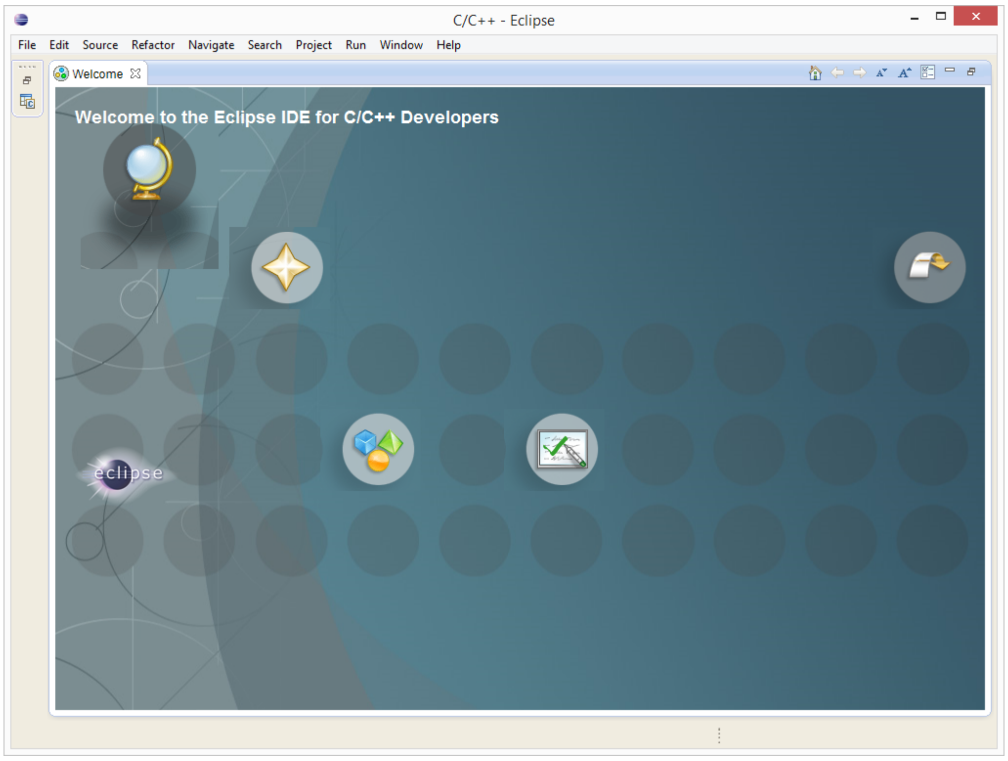
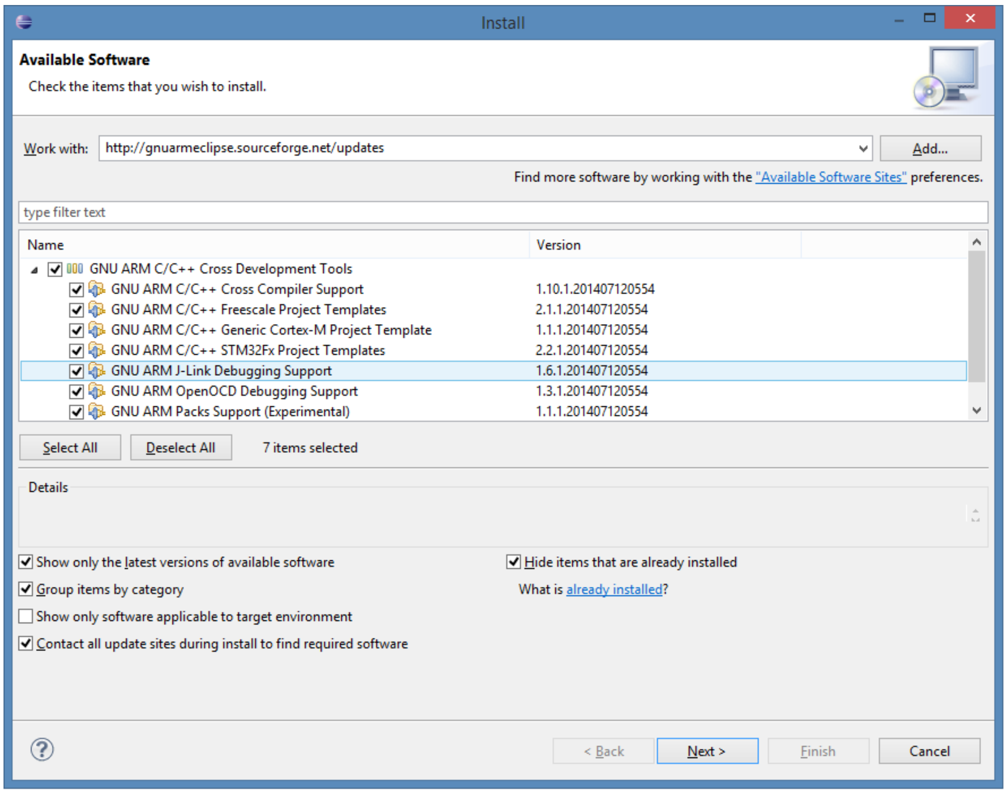

C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome to the Eclipse IDE for C/C++ Developers
eclipse

Figure 6: Eclipse welcome window

Now you have to install some plug-ins.

6.2 Install the GNU ARM Development support

Go to *Help* → *Install new software...* window and paste this URL *"http://gnuarmeclipse.sourceforge.net/updates"* to the *"Work with"* line (Figure 7). Install the plug-in.



Available Software

Check the items that you wish to install.

Work with: http://gnuarmeclipse.sourceforge.net/updates

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☑ GNU ARM C/C++ Cross Development Tools	
☑ GNU ARM C/C++ Cross Compiler Support	1.10.1.201407120554
☑ GNU ARM C/C++ Freescale Project Templates	2.1.1.201407120554
☑ GNU ARM C/C++ Generic Cortex-M Project Template	1.1.1.201407120554
☑ GNU ARM C/C++ STM32Fx Project Templates	2.2.1.201407120554
☑ GNU ARM J-Link Debugging Support	1.6.1.201407120554
☑ GNU ARM OpenOCD Debugging Support	1.3.1.201407120554
☑ GNU ARM Packs Support (Experimental)	1.1.1.201407120554

7 items selected

Details

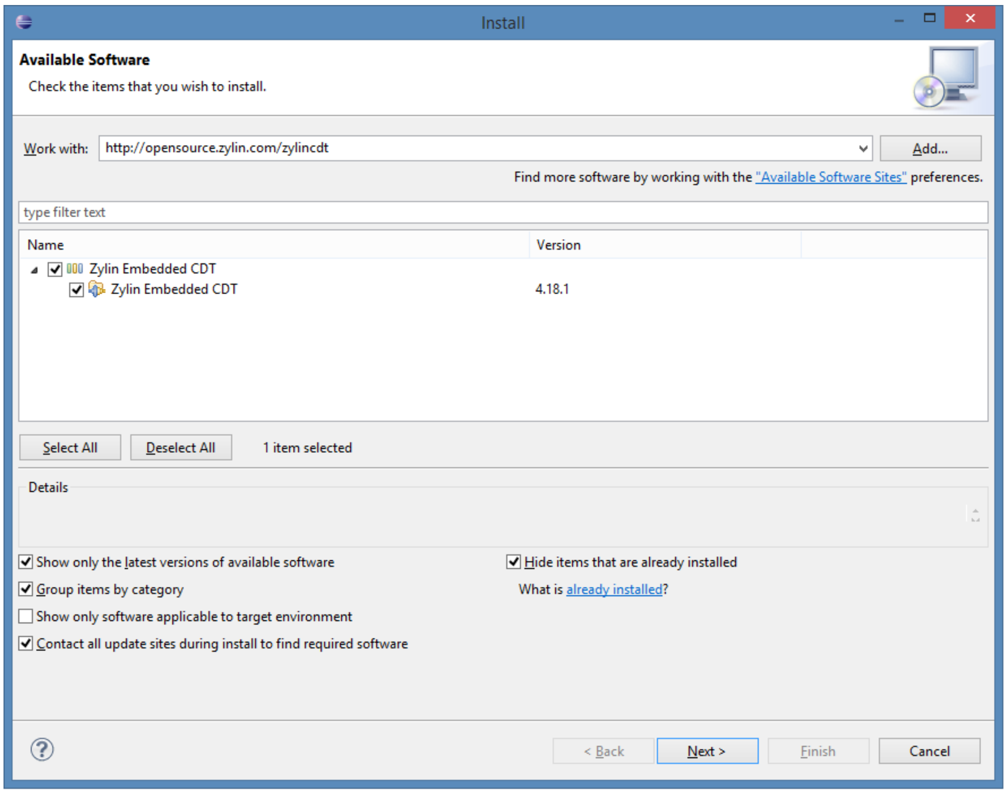
☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

Figure 7: Install GNU ARM development support

This plug-in adds automatic makefile creation capabilities and tool-chain preconfiguration to Eclipse. It supports various opensource tool-chains.

6.3 Install the Zylin CDT plug-in

To install the Zylin CDT plug-in go again to the *"Install new software"* window and paste this URL *"http://opensource.zylin.com/zylincdt"* select the plug-in and install it (Figure 8).



Install

Available Software

Check the items that you wish to install.

Work with: http://opensource.zylin.com/zylincdt

Find more software by working with the "Available Software Sites" preferences.

Name	Version
☑ Zylin Embedded CDT	
☑ Zylin Embedded CDT	4.18.1

1 item selected

Details

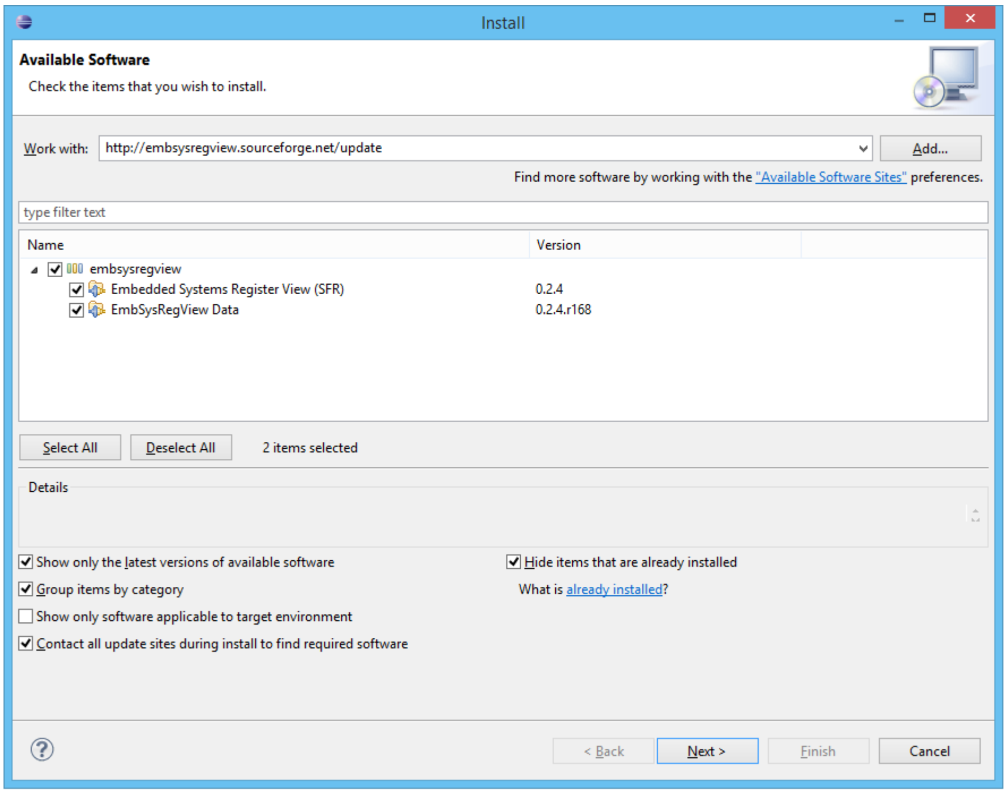
☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

Figure 8: Install the Zylin Embedded CDT

The Zylin CDT plug-in enables Eclipse to communicate with GDB debugger.

6.4 Install the EmbSys peripheral register view plug-in

To install the EmbSys peripheral register view plug-in go to the *"Install new software"* window and paste this URL *"http://embsysregview.sourceforge.net/update"* select the plug-in and install it (Figure 9).



Install

Available Software

Check the items that you wish to install.

Work with: http://embsysregview.sourceforge.net/update

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☑ embsysregview	
☑ Embedded Systems Register View (SFR)	0.2.4
☑ EmbSysRegView Data	0.2.4.r168

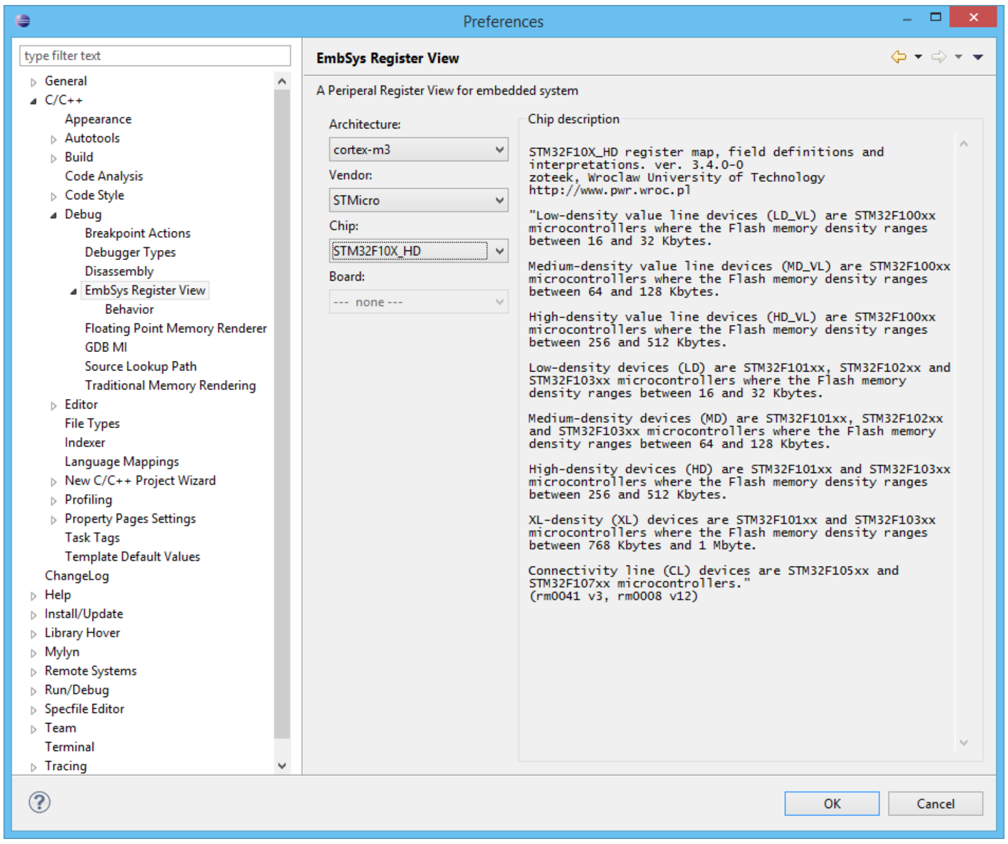
2 items selected

Details

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

Figure 9: EmbSys Register View plug-in installation

After installation go to *Window* → *Preferences* → *C/C++* → *Debug* → *EmbSys* Register view to setup the plug-in. For *Architecture* select *cortex m3*, for *Vendor* select *STMicro* and for *Chip* select *STM32F10x_HD_VL* click *OK* (Figure 10). To add the plug-in into a view go to *Window* → *Show View* → *Other...* → *Debug* and select *EmbSys Register*. Now you can view and change the peripheral registers of the microcontroller when debugging (Figure 11).



EmbSys Register View

A Periperal Register View for embedded system

Architecture: cortex-m3

Vendor: STMicro

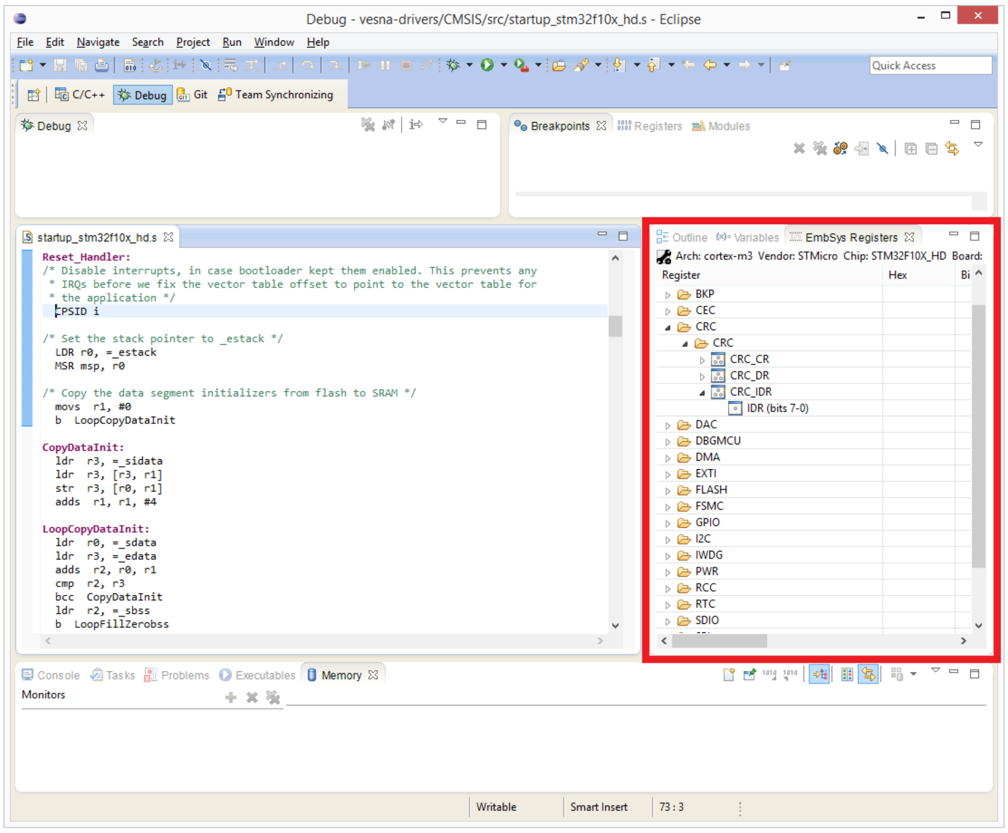
Chip: STM32F10X_HD

Board: --- none ---

Chip description

```
STM32F10X_HD register map, field definitions and
interpretations. ver. 3.4.0-0
zoteek, Wroclaw University of Technology
http://www.pwr.wroc.pl

"Low-density value line devices (LD_VL) are STM32F100xx
microcontrollers where the Flash memory density ranges
between 16 and 32 Kbytes.

Medium-density value line devices (MD_VL) are STM32F100xx
microcontrollers where the Flash memory density ranges
between 64 and 128 Kbytes.

High-density value line devices (HD_VL) are STM32F100xx
microcontrollers where the Flash memory density ranges
between 256 and 512 Kbytes.

Low-density devices (LD) are STM32F101xx, STM32F102xx and
STM32F103xx microcontrollers where the Flash memory
density ranges between 16 and 32 Kbytes.

Medium-density devices (MD) are STM32F101xx, STM32F102xx
and STM32F103xx microcontrollers where the Flash memory
density ranges between 64 and 128 Kbytes.

High-density devices (HD) are STM32F101xx and STM32F103xx
microcontrollers where the Flash memory density ranges
between 256 and 512 Kbytes.

XL-density (XL) devices are STM32F101xx and STM32F103xx
microcontrollers where the Flash memory density ranges
between 768 Kbytes and 1 Mbyte.

Connectivity line (CL) devices are STM32F105xx and
STM32F107xx microcontrollers."
(rm0041 v3, rm0008 v12)
```

Figure 10: EmbSys Register View setup


Debug - vesna-drivers/CMSIS/src/startup_stm32f10x_hd.s - Eclipse

startup_stm32f10x_hd.s

```
Reset_Handler:
/* Disable interrupts, in case bootloader kept them enabled. This prevents any
 * IRQs before we fix the vector table offset to point to the vector table for
 * the application */
  CPSID i

/* Set the stack pointer to _estack */
  LDR r0, =_estack
  MSR msp, r0

/* Copy the data segment initializers from flash to SRAM */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, =_edata
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInit
  ldr  r2, =_sbss
  b  LoopFillZerobss
```

EmbSys Registers

Arch: cortex-m3 Vendor: STMicro Chip: STM32F10X_HD Board:

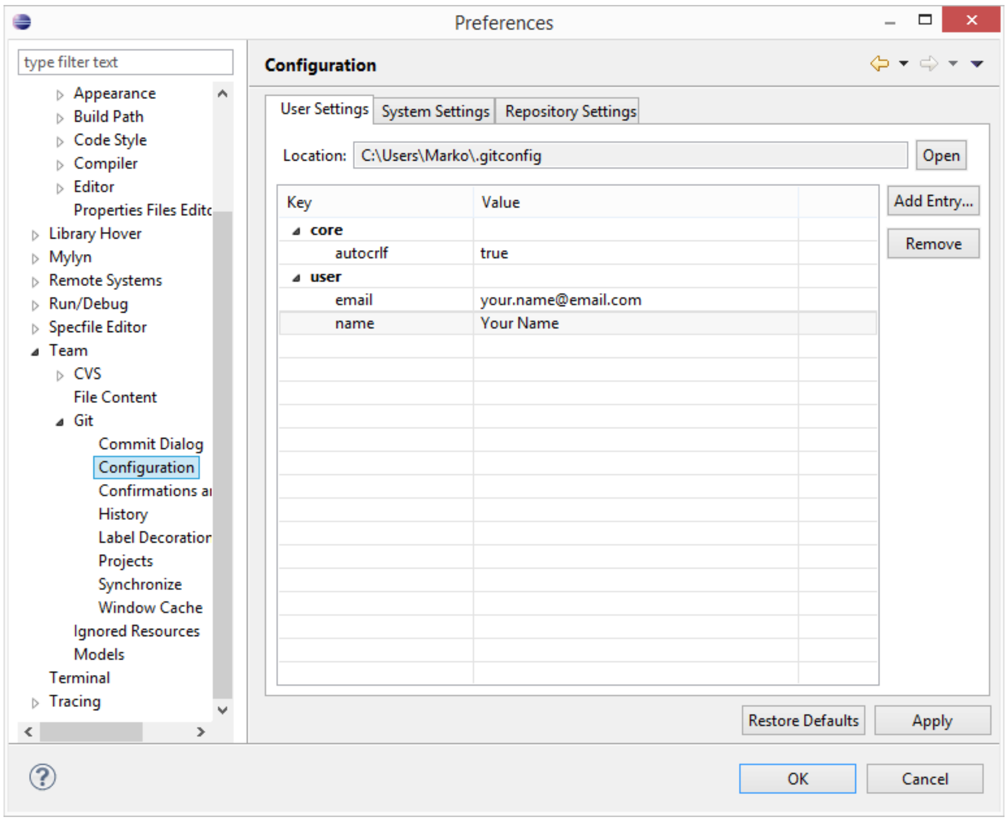
```
Register                      Hex    Bi
▷ BKP
▷ CEC
◢ CRC
    ◢ CRC
        ▷ CRC_CR
        ▷ CRC_DR
        ◢ CRC_IDR
            IDR (bits 7-0)
▷ DAC
▷ DBGMCU
▷ DMA
▷ EXTI
▷ FLASH
▷ FSMC
▷ GPIO
▷ I2C
▷ IWDG
▷ PWR
▷ RCC
▷ RTC
▷ SDIO
```

Writable   Smart Insert   73 : 3

Figure 11: EmbSys Register view (red square)

6.5 Configure Eclipse Git client

A Git client is already included in the base installation of Eclipse, you just need to configure it. Open *Preferences* and navigate to *Team* → *Git* → *Configuration* and add two new keys, email and name (Figure 12). You can use your Github credentials.



Location: C:\Users\Marko\.gitconfig

Key	Value
core	
autocrlf	true
user	
email	your.name@email.com
name	Your Name

Figure 12: Eclipse Git client configuration

7 Connection of the debugger interface and installation of drivers

In this section the installation of the ST-LINK/V2 hardware debugger interface is explained.

1. First go to `http://www.st.com/web/catalog/tools/FM146/CL1984/SC724/SS1677/PF251168`, scroll down to the *"Related Tools and Software"* section and download the appropriate USB driver for your Windows version (STSW-LINK003 for Win 7 or STSW-LINK006 for Win 8). Unzip the downloaded file.

2. Now connect the debugger interface ST-LINK/V2 to the PC. Windows will not find the correct drivers for the debugger interface. Go to *"Device Manager"* and install the drivers manually.

3. Go to *"Other devices"* tab and look for an unknown device named *"STM32 STLink"* (Figure 13).

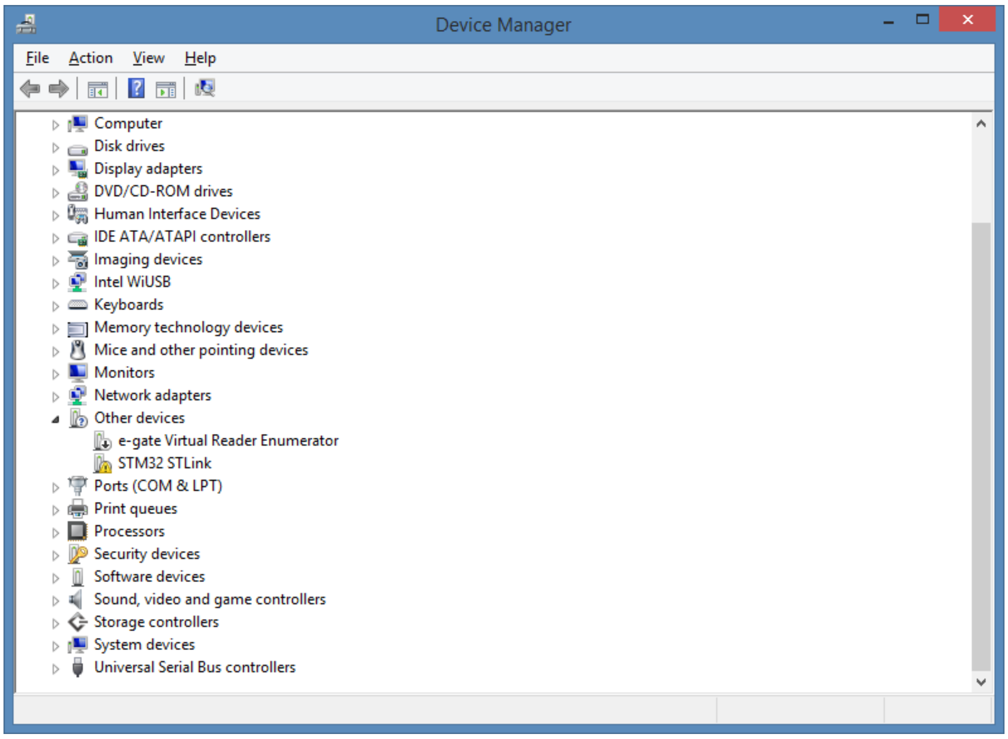
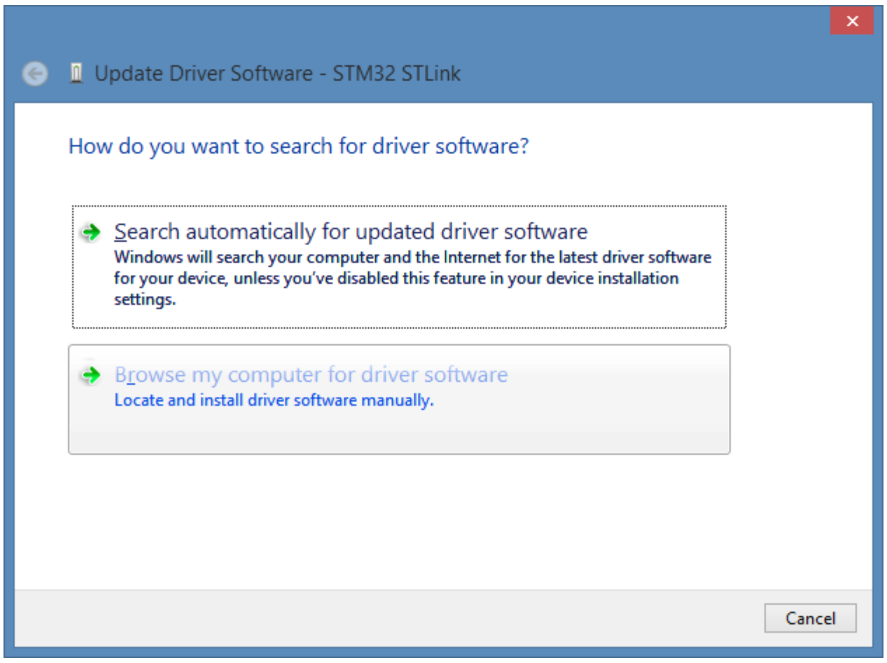


Figure 13: Device Manager when STLink is connected for the first time

4. Right-click on it and click *"Update Driver Software..."*

5. In the new window click on *"Browse my computer for driver software"* (Figure 14)


Update Driver Software - STM32 STLink

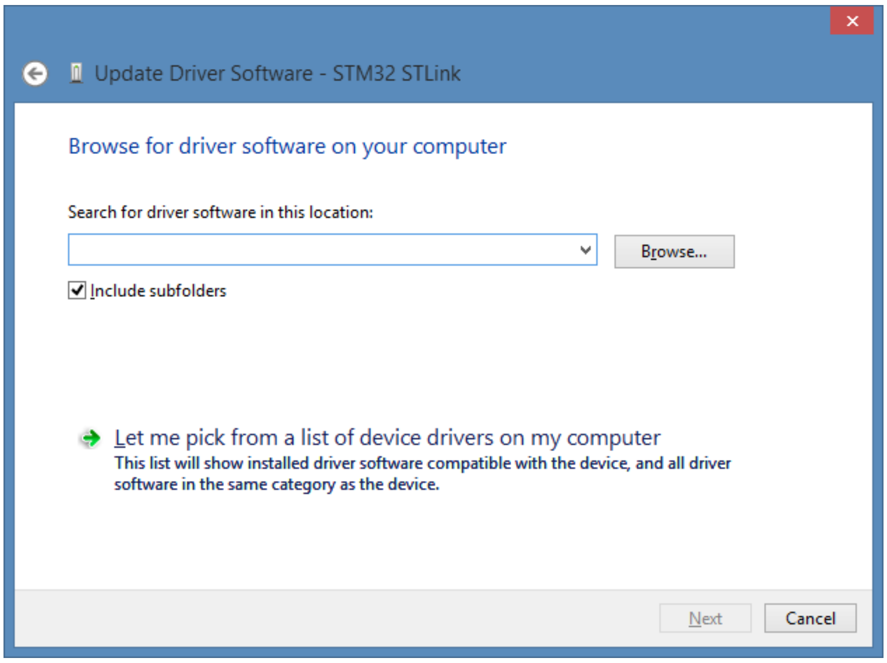
How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

Browse my computer for driver software
Locate and install driver software manually.

Figure 14: Select method to find driver for STLink

6. Browse to location where you've unzip the previously downloaded driver (Figure 15). Click next and the installation will start.


Update Driver Software - STM32 STLink

Browse for driver software on your computer

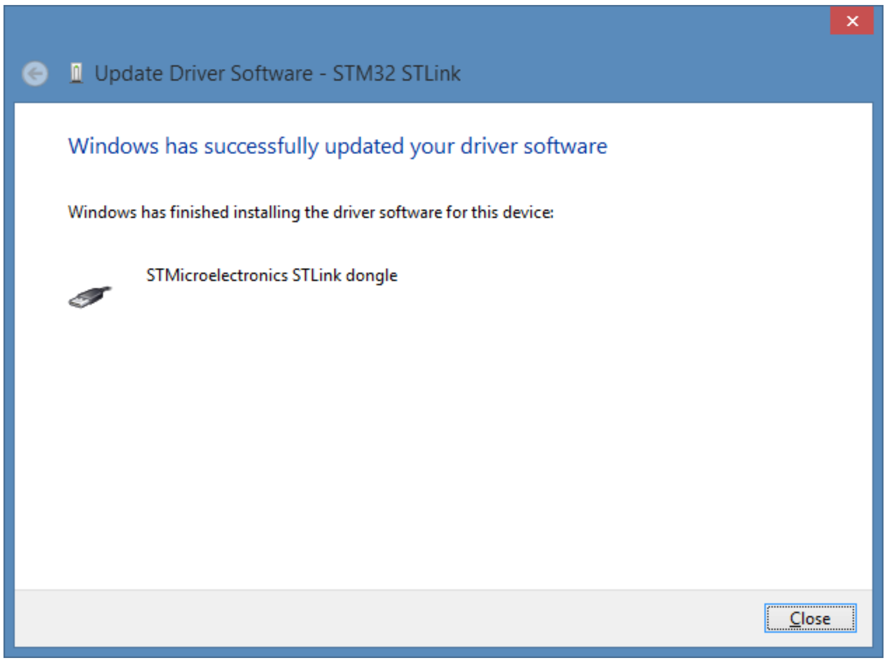
Search for driver software in this location:

☑ Include subfolders

Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Figure 15: Browse to driver location

7. The driver is now installed (Figure 16).


Update Driver Software - STM32 STLink

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

STMicroelectronics STLink dongle

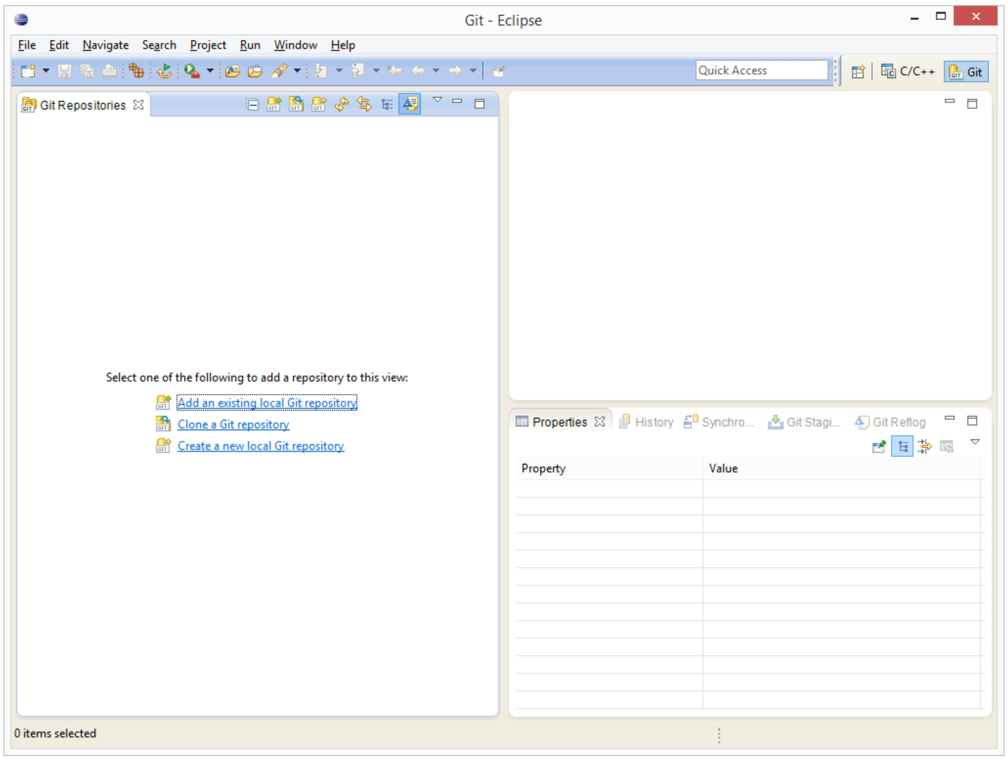
Figure 16: Driver installation complete

8 Download VESNA repository, Build and Debug

This chapter explains how to configure the Git plug-in, download the *"vesna-drivers"* repository, compile the source code, upload the build binary file to SNC and start a debug session. A Github (`https://github.com/`) account and access to the *"vesna-drivers"* repository is a prerequisite for the completion of this section. This guide does not cover the proper usage of Git or Github.

8.1 Download VESNA repository from Github

In Eclipse open the *"Git"* perspective. You can see that there are no available repositories. You can add the *"vesna-drivers"* repository by clicking on *"Clone a Git repository"* (Figure 17).



Select one of the following to add a repository to this view:

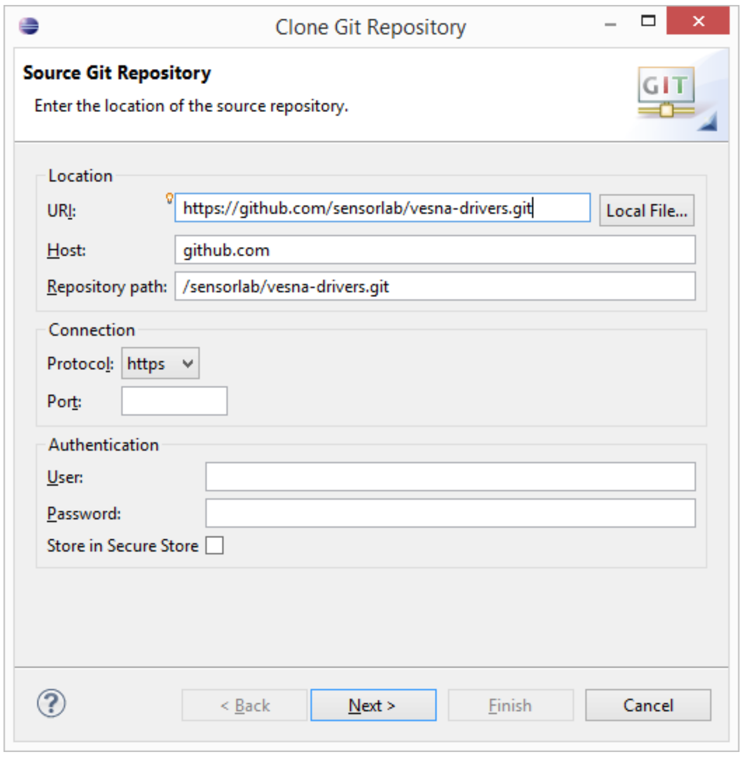
Add an existing local Git repository

Clone a Git repository

Create a new local Git repository

Figure 17: Git perspective in Eclipse

Copy the repository URI: *https://github.com/sensorlab/vesna-drivers.git* to the URI field (Figure 18). The other filed in Location frame should auto-complete. Fill in *username* and *password* of your Github account in the *"Authentication"* frame. Click Next.


Clone Git Repository

Source Git Repository

Enter the location of the source repository.

Location

URI: https://github.com/sensorlab/vesna-drivers.git

Host: github.com

Repository path: /sensorlab/vesna-drivers.git

Connection

Protocol: https

Port:

Authentication

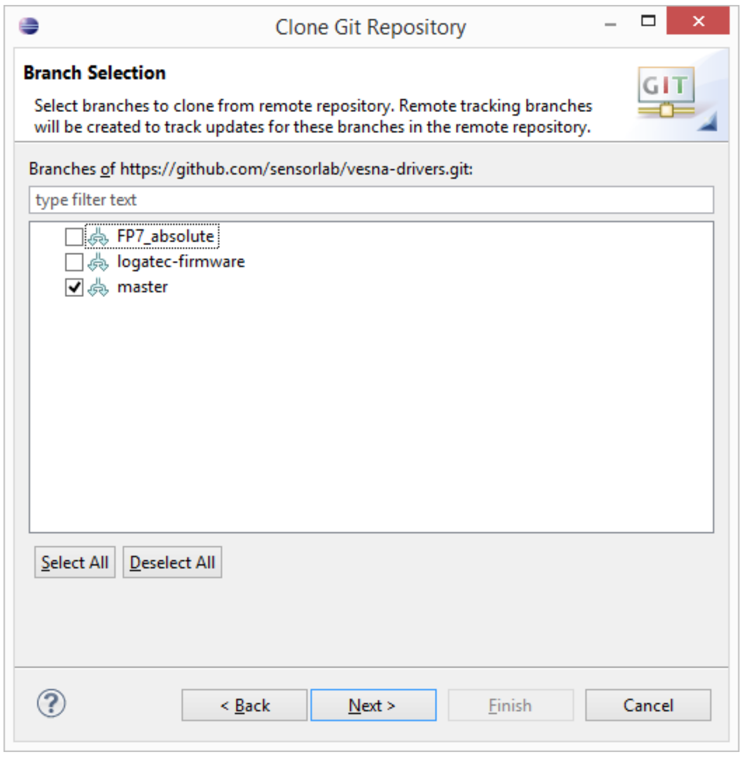
User:

Password:

Store in Secure Store ☐

Figure 18: Git repository location and credentials

Now you will be presented with a branch selection window. Select only the *"master"* branch as it is the most current (Figure 19). Click Next.


Clone Git Repository

Branch Selection

Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.

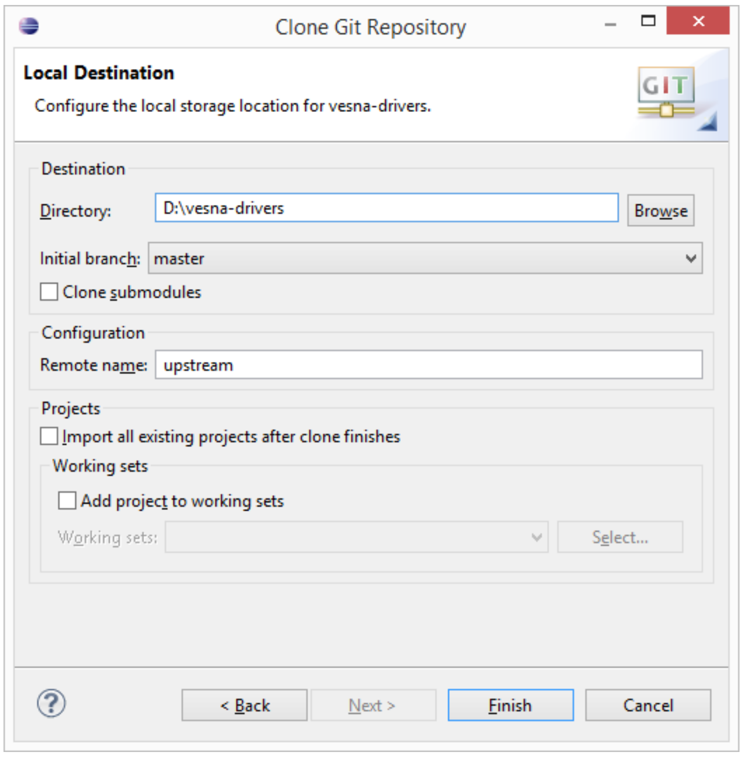
Branches of https://github.com/sensorlab/vesna-drivers.git:

type filter text

- [ ] FP7_absolute
- [ ] logatec-firmware
- [x] master

Figure 19: Git repository branch selection

Now you must select the destination where the files will be stored. This can be any location but the path may not contain spaces and non-US characters (Figure 20). Change the *"Remote name"* to *"upstream"*. Click Next to start the cloning process, this may take a minute or two.


Clone Git Repository

Local Destination

Configure the local storage location for vesna-drivers.

Destination

Directory: D:\vesna-drivers

Initial branch: master

☐ Clone submodules

Configuration

Remote name: upstream

Projects

☐ Import all existing projects after clone finishes

Working sets

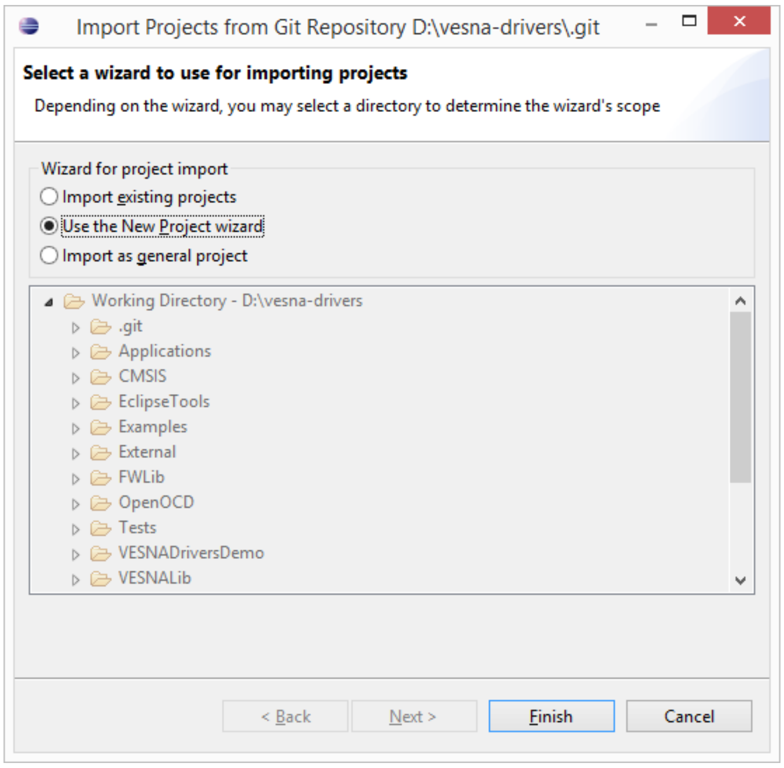
☐ Add project to working sets

Working sets:

Figure 20: Git repository local destination

8.2 Import project

Now that the repository has been cloned you have to import it to Eclipse workspace. Right-click on the *"vesna-drivers"* repository in *Git* perspective and select *"Import Projects..."*. Select *"Use the New project wizard"* and click *Finish* (Figure 21).


Import Projects from Git Repository D:\vesna-drivers\.git

Select a wizard to use for importing projects

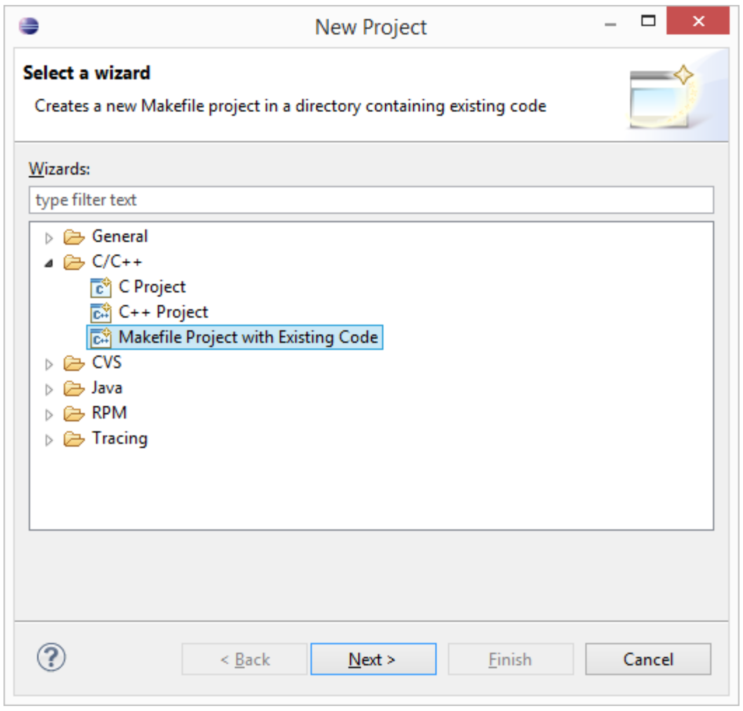
Depending on the wizard, you may select a directory to determine the wizard's scope

Wizard for project import
- ( ) Import existing projects
- (●) Use the New Project wizard
- ( ) Import as general project

```
Working Directory - D:\vesna-drivers
  .git
  Applications
  CMSIS
  EclipseTools
  Examples
  External
  FWLib
  OpenOCD
  Tests
  VESNADriversDemo
  VESNALib
```

Figure 21: Project import wizard selection

Now a new window opens. Select *"C/C++ → Makefile project with Existing Code"* and click Next (Figure 22).



New Project

Select a wizard

Creates a new Makefile project in a directory containing existing code

Wizards:

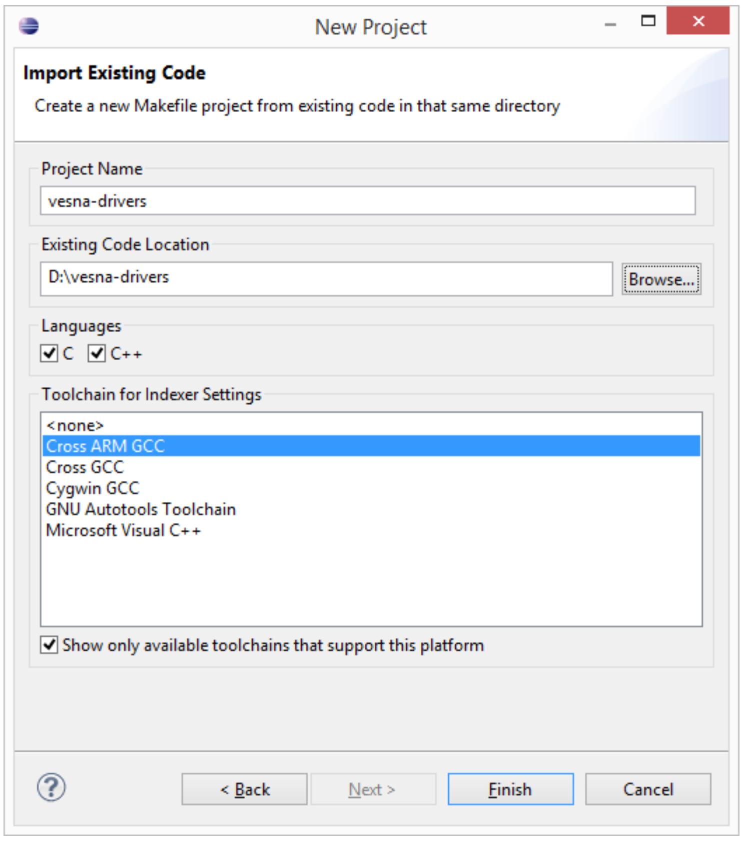
type filter text

- General
- C/C++
  - C Project
  - C++ Project
  - Makefile Project with Existing Code
- CVS
- Java
- RPM
- Tracing

< Back | Next > | Finish | Cancel

Figure 22: New project type selection

Name the project *"vesna-drivers"*, browse to the code location, select both C and C++ as the Language and select *"Cross ARM GCC"* as the tool-chain (Figure 23). The code is located in the directory you have selected as the Git repository location. Click *Finish*.


New Project

Import Existing Code

Create a new Makefile project from existing code in that same directory

Project Name: vesna-drivers

Existing Code Location: D:\vesna-drivers

Languages: ☑ C ☑ C++

Toolchain for Indexer Settings:
- <none>
- Cross ARM GCC
- Cross GCC
- Cygwin GCC
- GNU Autotools Toolchain
- Microsoft Visual C++

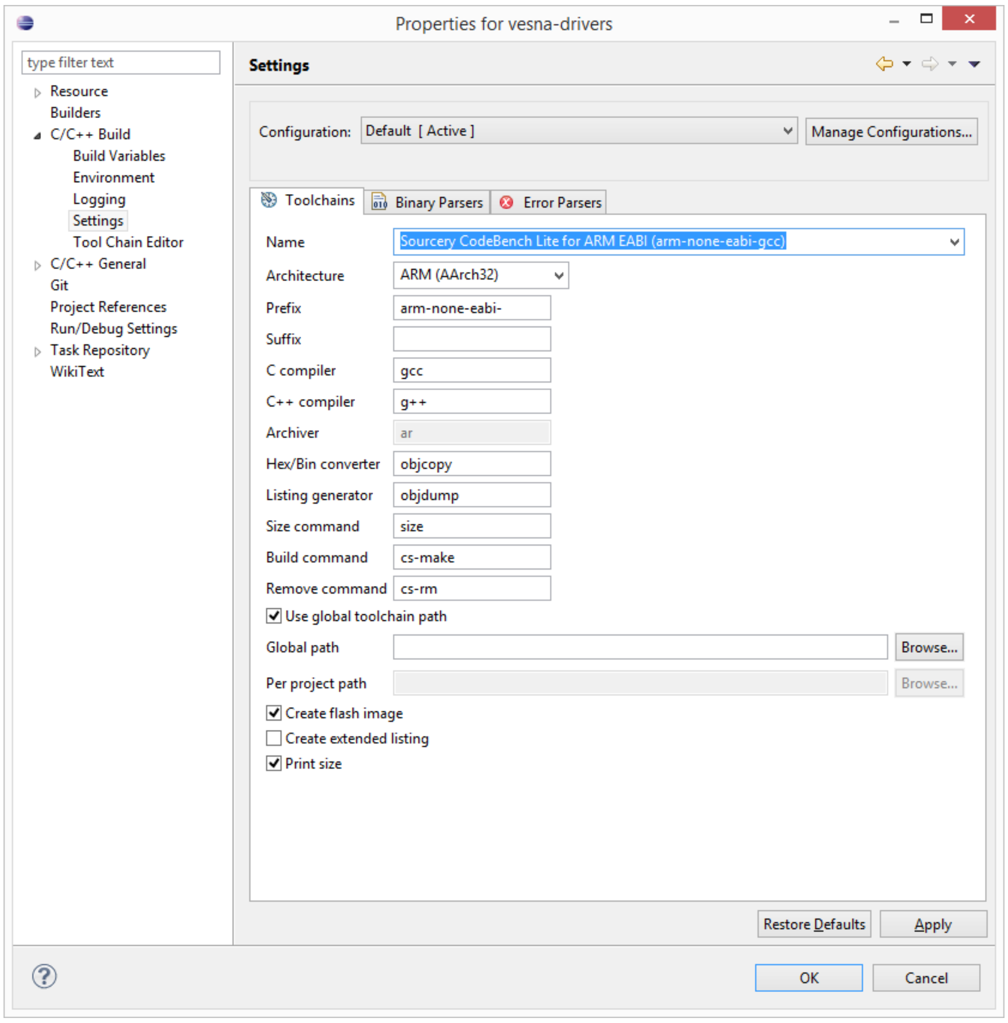
☑ Show only available toolchains that support this platform

Figure 23: Import existing code window

Switch to the *"C/C++"* perspective. You can now see the project structure in *"Project Explorer"*.

8.3 Configure the project

In *"Project Explorer"* right click on the top folder (vesna-drivers) and lef-click on *"Properties"*. Go to *C/C++ Build* → *Settings* and change the *"Name"* field under *"Toolchains"* to *"Sourcery CodeBench Lite for ARM EABI (arm-none-eabi-gcc)"* (Figure 24).


Properties for vesna-drivers

Settings

Configuration: Default [Active]

Toolchains

Name	Sourcery CodeBench Lite for ARM EABI (arm-none-eabi-gcc)
Architecture	ARM (AArch32)
Prefix	arm-none-eabi-
Suffix	
C compiler	gcc
C++ compiler	g++
Archiver	ar
Hex/Bin converter	objcopy
Listing generator	objdump
Size command	size
Build command	cs-make
Remove command	cs-rm

☑ Use global toolchain path

Global path

Per project path

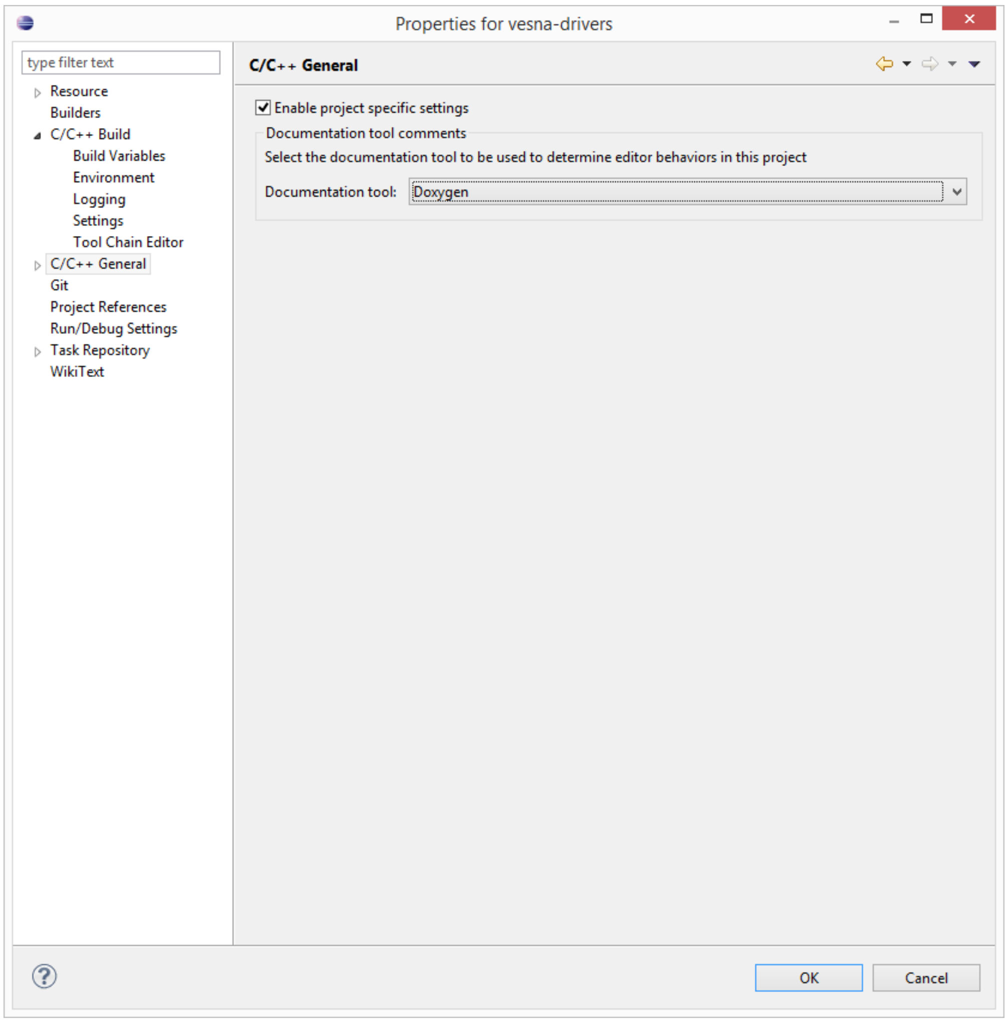
☑ Create flash image

☐ Create extended listing

☑ Print size

Figure 24: Project tool-chain settings

Go to *"C/C++ General"* and tick the *"Enable project specific settings"* box and select *"Doxygen"* as the documentation tool (Figure 25).



☑ Enable project specific settings

Documentation tool comments

Select the documentation tool to be used to determine editor behaviors in this project

Documentation tool: Doxygen

Figure 25: Documentation tool settings

The basic project configuration is now complete. When working with Git the project configuration files are ignored and are not changed when you switch between different commits or branches. In general project import and configuration must be done only when the repository is cloned for the first time.

8.4 Compile the source

Go to *"C/C++"* perspective and in *"Project Explorer"* open the *"VESNADriversDemo"* folder. This is the demo project you will compile and upload to SNC (Figure 26).

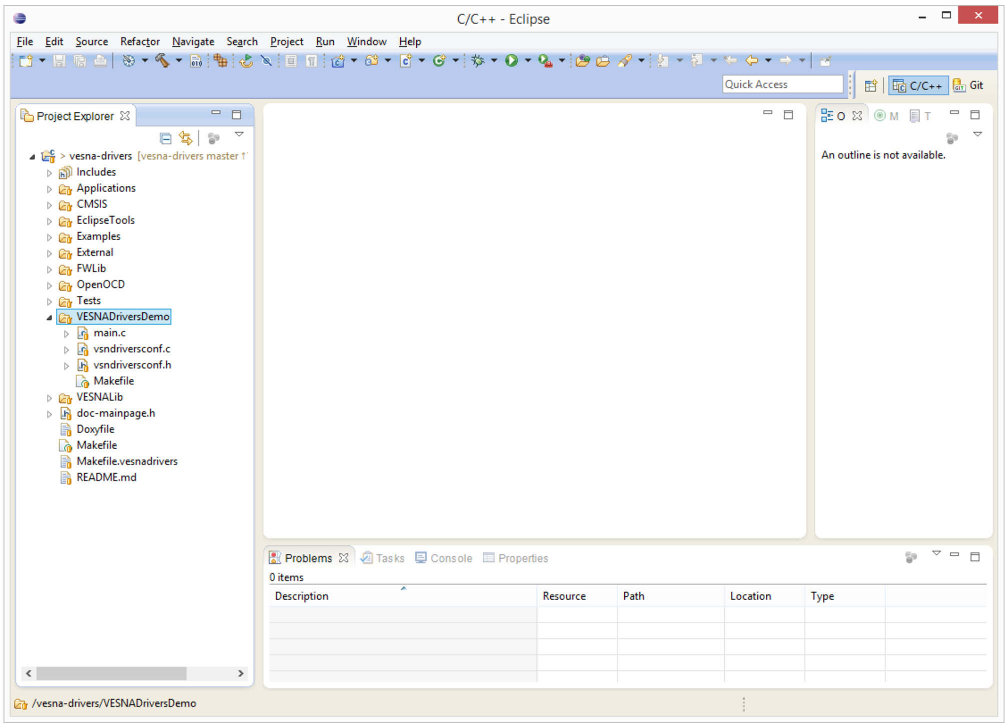
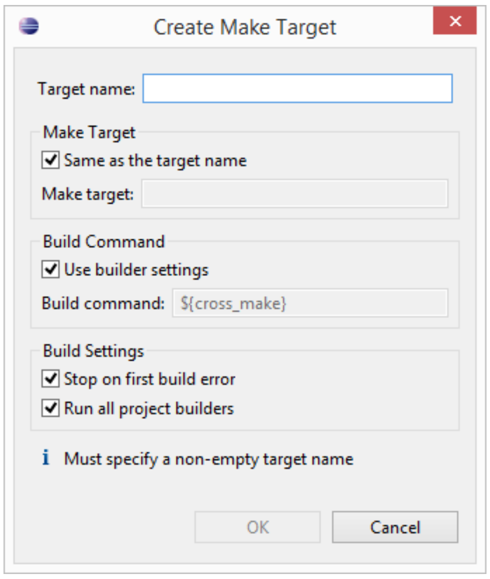


Figure 26: C/C++ perspective demo project

The makefile system configures the code build process and all the needed files are build automatically. So that the build system knows what to build and how you need to pass it the correct make targets. The two basic targets are *"all"* and *"clean"* which build the project or remove all the build process files respectively. Each project has at least two files *"main.c"* and *"Makefile"*. The file *"main.c"* is the source of the main application code and the file *"Makefile"* is part of the build system. To compile the code you have to append make targets to the *"Makefile"*. You do this by right-clicking on the *"Makefile"* and left-click on *Make Targets* → *Create...* and fill in the *"Target name"* first *"all"* and then do the same thing for *"clean"* (Figure 27).



Create Make Target

Target name:

Make Target

☑ Same as the target name

Make target:

Build Command

☑ Use builder settings

Build command: ${cross_make}

Build Settings

☑ Stop on first build error

☑ Run all project builders

i Must specify a non-empty target name

Figure 27: Create make target

Find the *"Make Target"* tab on the right side (Figure 28). The folder structure is the same as in *"Project Explorer"* but there are no files only make targets. To see the make targets you have just created open the *"VESNADriversDemo"* folder. Double-click on the *"all"* make target and the project will start building. You can follow the build process in the *"Console"* tab at the bottom of the Eclipse window (Figure 29).

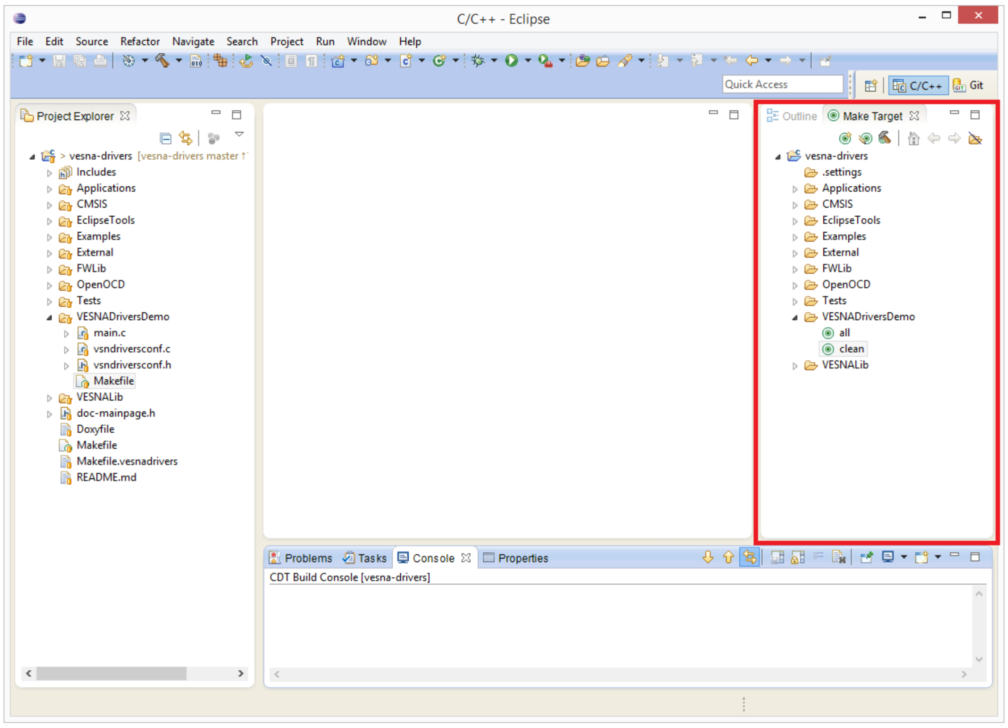
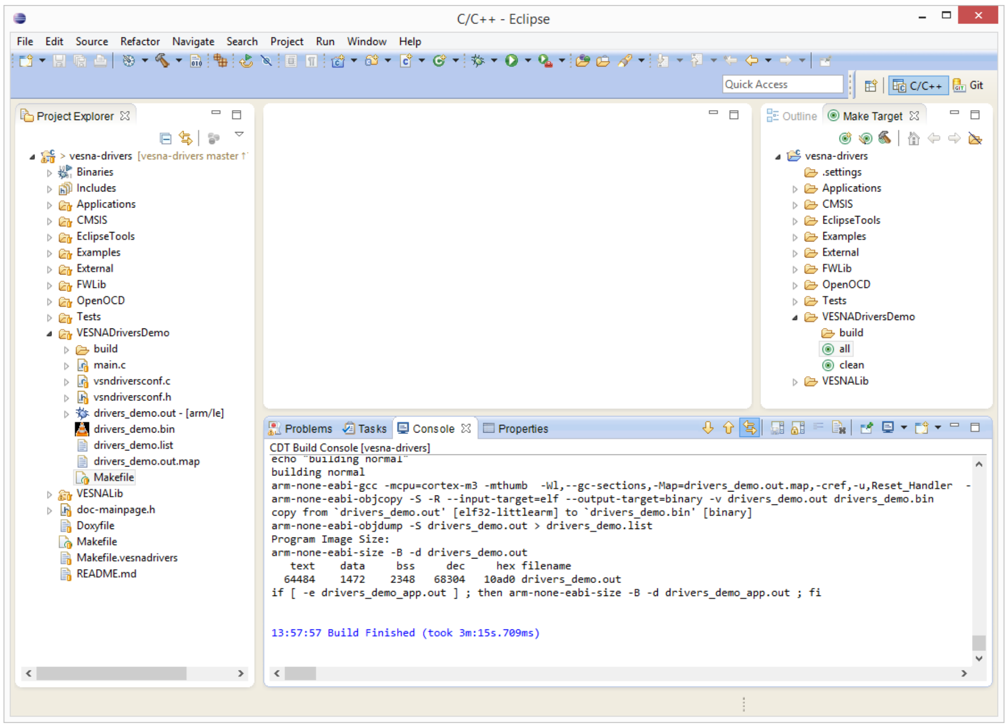


Figure 28: "Make Target" tab (in red square)



```
echo "building normal"
building normal
arm-none-eabi-gcc -mcpu=cortex-m3 -mthumb  -Wl,--gc-sections,-Map=drivers_demo.out.map,-cref,-u,Reset_Handler  -
arm-none-eabi-objcopy -S -R --input-target=elf --output-target=binary -v drivers_demo.out drivers_demo.bin
copy from `drivers_demo.out' [elf32-littlearm] to `drivers_demo.bin' [binary]
arm-none-eabi-objdump -S drivers_demo.out > drivers_demo.list
Program Image Size:
arm-none-eabi-size -B -d drivers_demo.out
   text    data     bss     dec     hex filename
  64484    1472    2348   68304   10ad0 drivers_demo.out
if [ -e drivers_demo_app.out ] ; then arm-none-eabi-size -B -d drivers_demo_app.out ; fi


13:57:57 Build Finished (took 3m:15s.709ms)
```

Figure 29: Building the code

9 Upload the project and start debug

Now connect the SNC board to the debugger and power supply. Click on the arrow in the *"External Tools"* icon and select *"OpenOCD-STLinkV2 VESNA Drivers Debug"* (Figure 30). This will start OpenOCD, connect STLink V2 debugger to OpenOCD and configure SNC for debugging.

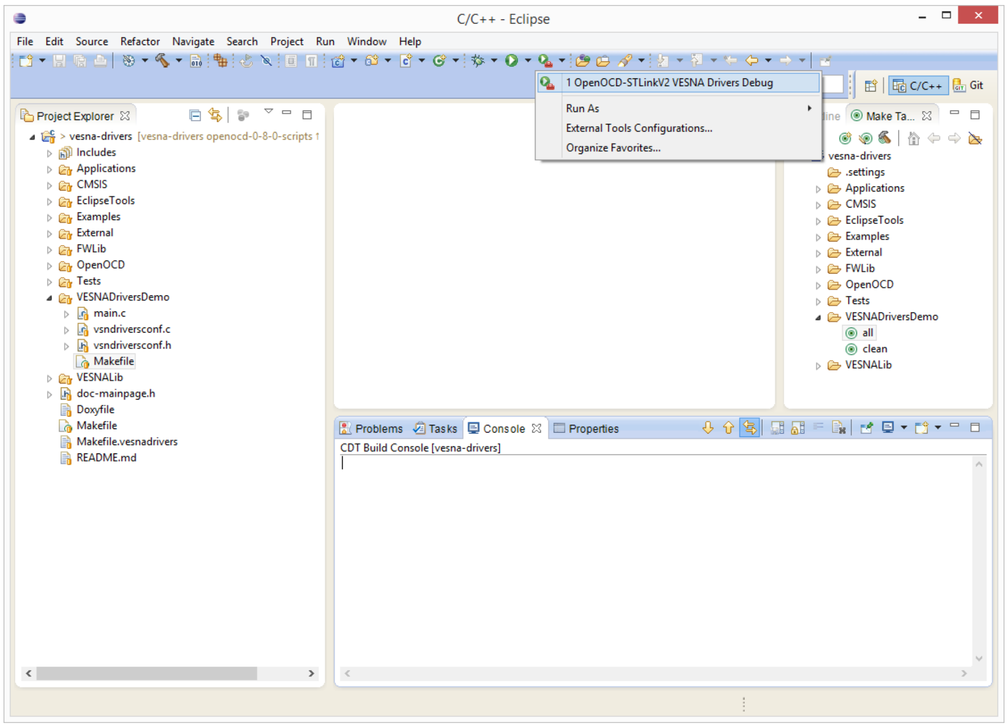
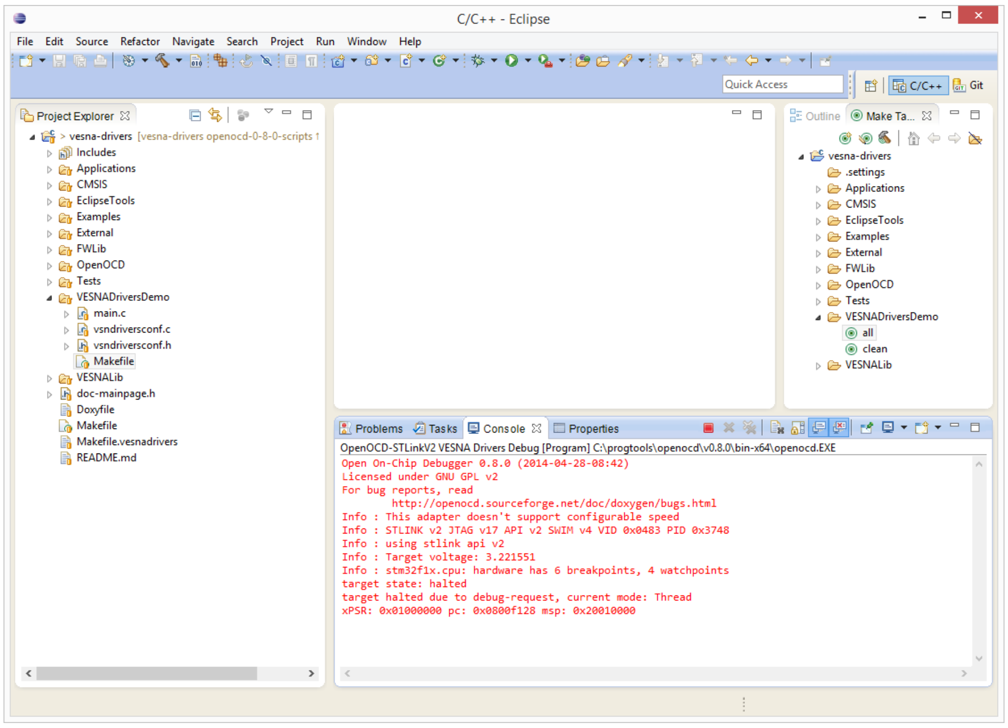


Figure 30: External tool selection

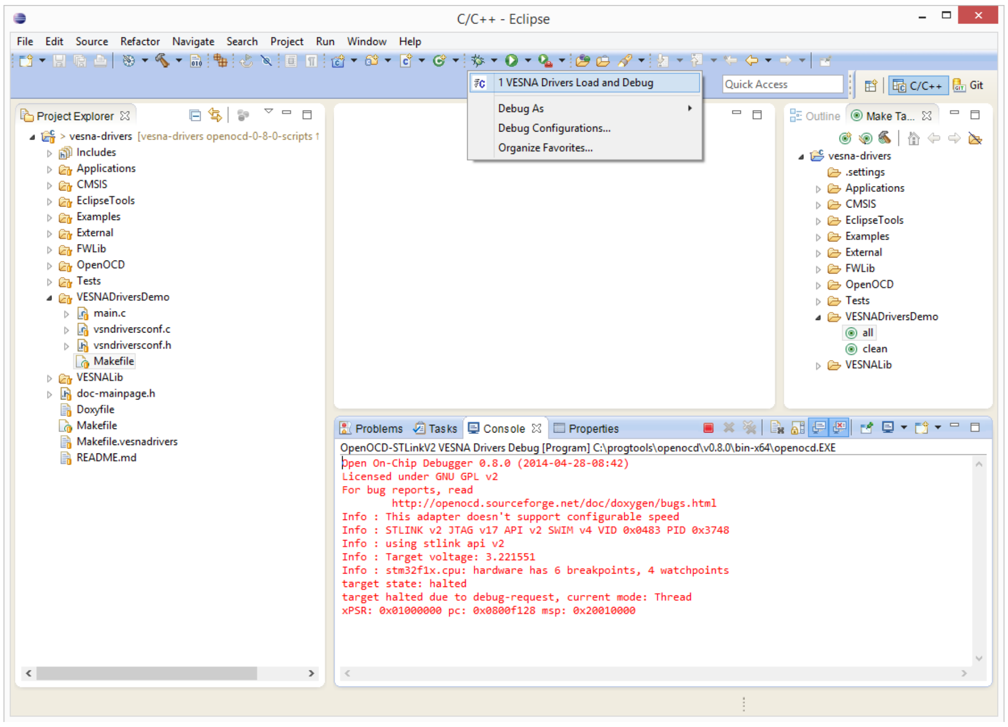
If all is working you should see something like this in the *"Console"* window on Figure 31.



```
OpenOCD-STLinkV2 VESNA Drivers Debug [Program] C:\progtools\openocd\v0.8.0\bin-x64\openocd.EXE
Open On-Chip Debugger 0.8.0 (2014-04-28-08:42)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : This adapter doesn't support configurable speed
Info : STLINK v2 JTAG v17 API v2 SWIM v4 VID 0x0483 PID 0x3748
Info : using stlink api v2
Info : Target voltage: 3.221551
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x0800f128 msp: 0x20010000
```

Figure 31: External tool running

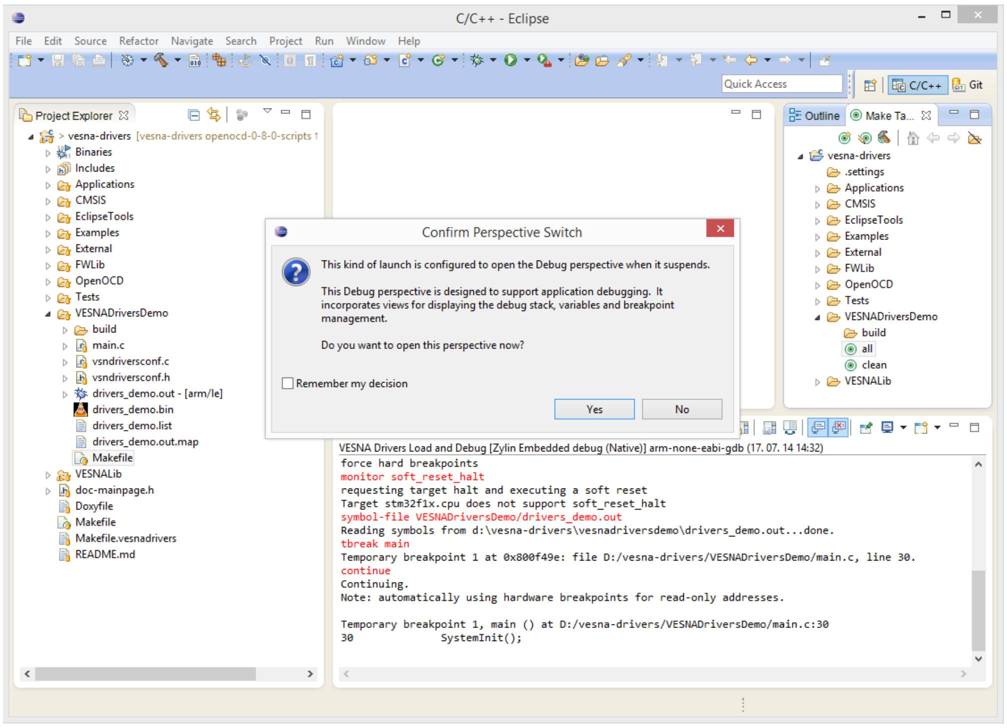
This means that the debugger hardware is found and it is connected to SNC. Now start the debugger by clicking on the arrow in *"Debugger configuration"* icon and click on the *"VESNA Drivers Load and Debug"* (Figure 32). This will upload the software to SBC and start the debugger.



```
OpenOCD-STLinkV2 VESNA Drivers Debug [Program] C:\progtools\openocd\v0.8.0\bin-x64\openocd.EXE
Open On-Chip Debugger 0.8.0 (2014-04-28-08:42)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : This adapter doesn't support configurable speed
Info : STLINK v2 JTAG v17 API v2 SWIM v4 VID 0x0483 PID 0x3748
Info : using stlink api v2
Info : Target voltage: 3.221551
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x0800f128 msp: 0x20010000
```

Figure 32: Debugger configuration

You are prompted to switch the view to the *"Debug"* perspective, click *Yes* (Figure 33).



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

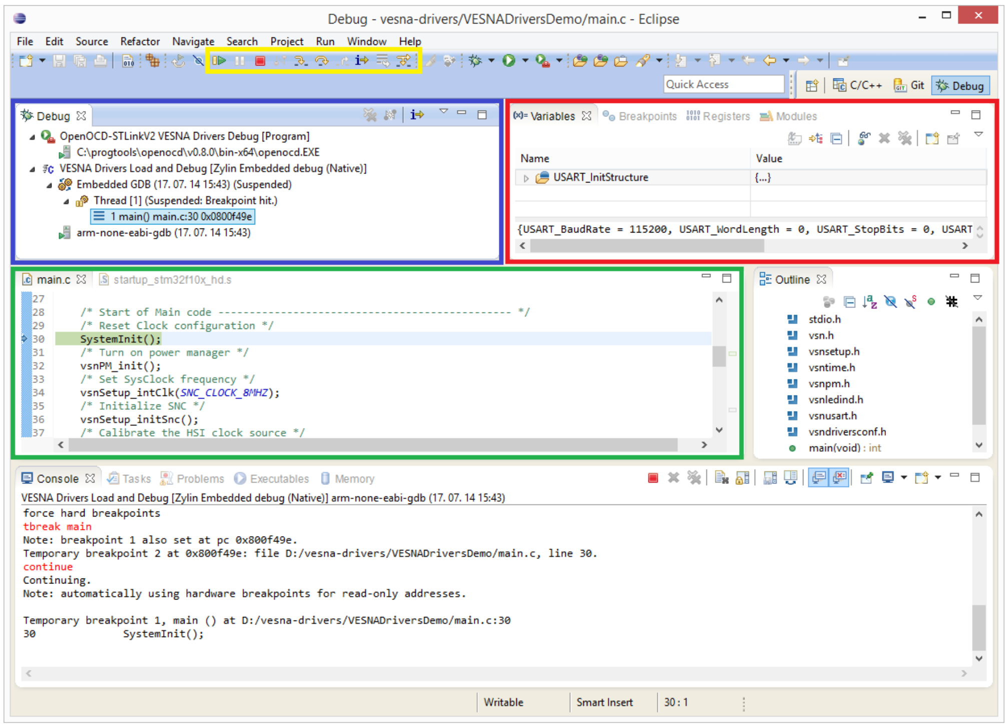
☐ Remember my decision

VESNA Drivers Load and Debug [Zylin Embedded debug (Native)] arm-none-eabi-gdb (17. 07. 14 14:32)

```
force hard breakpoints
monitor soft_reset_halt
requesting target halt and executing a soft reset
Target stm32f1x.cpu does not support soft_reset_halt
symbol-file VESNADriversDemo/drivers_demo.out
Reading symbols from d:\vesna-drivers\vesnadriversdemo\drivers_demo.out...done.
tbreak main
Temporary breakpoint 1 at 0x800f49e: file D:/vesna-drivers/VESNADriversDemo/main.c, line 30.
continue
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Temporary breakpoint 1, main () at D:/vesna-drivers/VESNADriversDemo/main.c:30
30              SystemInit();
```

Figure 33: Prompt to switch to "Debug" perspective

Now you should see something like Figure 34. The program is automatically paused at the first line of main. The current position of the program in the code can bee seen in the code tab (green rectangle). The pausing and running of the program can be controlled with the debug controls (yellow rectangle). Variables can be inspected in the *"Variables"* tab (red rectangle). Running debug tools (OpenOCD and GDB) can be seen in the *"Debug"* tab (blue frame). You can also see the current stack composition in the *"Debug"* window if you look under the *"Thread[1].."* line.



```
Debug
OpenOCD-STLinkV2 VESNA Drivers Debug [Program]
    C:\progtools\openocd\v0.8.0\bin-x64\openocd.EXE
VESNA Drivers Load and Debug [Zylin Embedded debug (Native)]
    Embedded GDB (17. 07. 14 15:43) (Suspended)
        Thread [1] (Suspended: Breakpoint hit.)
            1 main() main.c:30 0x0800f49e
    arm-none-eabi-gdb (17. 07. 14 15:43)
```

Name	Value
USART_InitStructure	{...}

```
{USART_BaudRate = 115200, USART_WordLength = 0, USART_StopBits = 0, USART
```

```c
    /* Start of Main code ------------------------------------------- */
    /* Reset Clock configuration */
    SystemInit();
    /* Turn on power manager */
    vsnPM_init();
    /* Set SysClock frequency */
    vsnSetup_intClk(SNC_CLOCK_8MHZ);
    /* Initialize SNC */
    vsnSetup_initSnc();
    /* Calibrate the HSI clock source */
```

```
stdio.h
vsn.h
vsnsetup.h
vsntime.h
vsnpm.h
vsnledind.h
vsnusart.h
vsndriversconf.h
main(void) : int
```

```
VESNA Drivers Load and Debug [Zylin Embedded debug (Native)] arm-none-eabi-gdb (17. 07. 14 15:43)
force hard breakpoints
tbreak main
Note: breakpoint 1 also set at pc 0x800f49e.
Temporary breakpoint 2 at 0x800f49e: file D:/vesna-drivers/VESNADriversDemo/main.c, line 30.
continue
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Temporary breakpoint 1, main () at D:/vesna-drivers/VESNADriversDemo/main.c:30
30              SystemInit();
```

Figure 34: Debug perspective, yellow frame: debug controls, blue frame: running debug tools and current stack position, red frame: local variables and green frame: current position in code

References

[1] FreeRTOS.org STM32 demo using Eclipse, OpenOCD and GCC.
`http://www.stf12.org/developers/ODeV.html`

[2] Kevin Townsend. Configuring Eclipse + OpenOCD + GCC to Debug NativeSample.
`http://msmicroframework.blogspot.com/2009/02/configuring-eclipse-openocd-gcc-to.html`

[3] Debugging STM32 Cortex-M3 microcontroller using Eclipse on Slackware.
`http://linuxfreak.pl/elektronika/debugging-stm32-cortex-m3-microcontroller-using-eclipse-on-slackware/`